DESIGN REPORT

Author: Le Minh Dang (mldang), Zi Xue Lim (zixue)

Overview:

Our project is separated into multiple individual files which are all written in Python3, except for *main.py, main_menu.py* and *util.py*, each file contains the component from the requirements of this project. In order to run the program, please run "**python3 main.py *database.db*".** The program will test if the file exists and make sure that ***.db*** is there. After that, the program will prompt for 2 options: Log In and Sign Up. When signing up, please make sure that the User ID is unique and only contains 4 characters. The program will warn if it is not entered properly.

After a successful login/signup session, the program will open the menu screen which the user will have 3 options: Post Question, Search for Post, And Log Out. For the first option, the program will pre-generate an Unique ID so users can focus on the title and the body of the question.

For the second option, you will enter the keyword of the program and it will return the result. Note that to make the terminal look properly, for each column, the program will check if the content is over 50 characters, it will truncate it to fit on the terminal. In order to select a post, the user is required to enter the number character based on *No.* Column. After the post is selected, for normal users, it will show that a user can post an answer to the post(if the post is a question, otherwise it will show error) and upvote the post or exit the selection menu to go back to the main menu. For privileged user (required the database must have a record of it, sign up will not create a privileged user), there are more options besides the normal user's option such as: edit post, mark as accepted or give a badge to poster (note that giving badge only allows if that poster has not get any badges for that day).

For the third option of the menu, the user can log out, and the program will close the database.

Detailed Description:

With the program being divided into multiple files, the components will be contained in these individual files:

- main.py
    - This is the main file of the entire program. This contains a function that can check for the third input and make sure that the database is entered correctly to open the connection and pass through the login phase
- Login.py
    - This contains the first component of the project: serving the purpose of making sure that only users within the database can access it. Also, it creates a sign up terminal, where users can enter their credentials. After it, it will return the user ID to feed to the main menu.
- main_menu.py
    - This serves as an interface between the database and the user after a successful login/signup session. It checks the privileged table in the database to see if the user is a privileged one or not. After checking, it will create a menu that serves the type of user as entered in the database.
- PostQuestions.py
    - This has the first functionality of the program: post a question. The function will automatically create a unique post ID using UUID and checking if the generated one is already in the posts table to make sure the uniqueness. Then it will prompt for the title and the body so that it can do an insert to the posts table and the questions table.
- SearchPost.py
    - This contains the second functionality based on the requirements. The function SearchMain will prompt for the keywords, then it will check and rank the post based on the number of keywords it finds in the post. This is done by searching each keyword in the database through the title, body and tags field and then using Union All to combine the results for all the keywords together. Following this, we would group by the post id and then reorder the query based on the highest count and limit it to 5. Then, the post id and poster will be returned to perform post action on it.
- giveAns.py
    - This contains the function giveAns which will provide a *Post Action-Answer* functionality. The function will check if the selected post ID is a question or not. If it is, then it will continue to auto generate a unique Post ID for preparing a new post. Then, it will prompt the user to enter the title, the body of a post similar when they post a question. But this time, it will insert into the answers table with the newly created post ID and the question post ID from the selected post at SearchMain.
- vote.py
    - This contains the function giveVotes which will provide a *Post Action-Vote* functionality. With the chosen Post ID from the SearchMain, it will check

the votes table to make sure that the user has already voted this post. If the user has not voted this post, it will create a vote number unique ID among other votes in that post. Then, it will update to the database with the necessary information on the vote table.

- acceptedAns.py
    - This file contains a function acceptedAns which has the *Post Action-Mark as Accepted Answer.* This functionality is only available for the privileged user. Before a post can be an accepted answer, the function will create a 2 stage check for the post: first, it will check if the post is an answer or not. If it is, it will have a second check to make sure that the question for this answer already has an accepted answer. If it is, the program will prompt the user if they want to change it. If they do, it will update the questions table with *theaid* with the selected post answer.
- giveBadge.py
    - This file contains the function giveBadge that has the *Post Action-Give a Badge*. This function will only be available to the privileged user. First, the function will check if the selected poster from the selected post already has a badge on that day. If the user already has, it will return back to the selection menu. Otherwise, it will prompt the user if they want to give this answer the badge. After the user put in the badge, it will check if the badge is available in the badges table. If it does, it will update the ubadge table with the necessary information from the prompted input with the poster and the date that they were given.
- giveTag.py
    - This file contains the function giveTag that has the *Post Action-Add a tag.* This function will only be available to the privileged user. It will prompt the user to input the name of the tag they will put. Then it will check if the tag exists (case insensitive). If it does not, it will insert the tag with the prompted tag name to the tags table. Otherwise, it will return to the selected post.
- EditPrivUser.py
    - This function will provide users that have the privilege role to edit posts after the post has been selected in the search post script. It is done by taking the PID passed to the function and using an sql query to get that row and output the information the user is going to edit. Then the user would need to provide inputs for the title and body of the post and then use the SQL update function to update the row with the same PID that was passed into the function.

Testing Strategy:

In terms of testing, we used the requirements error checking from CMPUT 291 Mini Project 1 specification as a backbone for our testing case. Also, we reused Vincent Wainman's SQL Data from Assignment 2 with a minor addition for the user's password and two privileged users for testing the components only available for the privileged user.

Because of our lack of knowledge for a good automated testing, we chose a traditional testing by manually testing the components. We manually prompted the program by the cases that we think could potentially break the program. Then, we launched a debugging session to figure out why the program was broken. We used VSCode pylance extension for python debugging session, vscode-sqlite as a GUI to see the change in the database if we launch a program. To make sure that the code can run on the lab machine, all of these plugins are installed on Ohaton server and we used vscode SSH Server to make Ohaton accessible from VSCode editor.

When testing the individual functionality, we created a function called func_test() (see *figure 1*) on that python file and prompt manually like a normal/privileged user and launched a debugging session.

```
def func_test():
    conn = sqlite3.connect('./test_data.db')
    print(str(SearchMain(conn.cursor())))
```

Figure 1: func_test() demo

Breaking Down Strategy:

We divide the task based on the specification posted on CMPUT 291. Also, we rank which task is the hardest, if it is hard, the person who did it will do less tasks than the other. All other tasks are democratically assigned based on what they would like to do. Some tasks are done by both because of their discoveries of bugs and contributing back to the project. We used Github as a way to make the contribution easier for the project (at the time of writing this, the repository is in private mode, if required, we can make it public after the submit time).
In the end, we decided to divide the task:

- **Zi Xue Lim**: *Post a question*; *Search for posts*; *Post Action - Edit*; *Login page*; *Main menu*
- **Le Minh Dang**: *Post Action - Answer*; *Post Action - Vote*; *Post Action - Mark as the Accepted*; *Post Action - Add a tag*; *Post Action - Add a badge*; *Main menu*; *Login Page*

We thought this would be the most efficient way to do it because some of these tasks would take longer than others to do.